

CONV-TTL-BLO HDL Guide

Gateway v1.02
August 7, 2013



Theodor-Adrian Stana (CERN/BE-CO-HT)

Revision history

Date	Version	Change
04-07-2013	0.1	First draft
26-07-2013	0.2	Second draft
07-08-2013	1.02	Added pulse rejection to <i>ctb_pulse_gen</i>

Contents

1	Introduction	1
2	FPGA Clocks	1
3	Reset generator	2
4	RTM detection	3
5	Bicolor LED controller	4
5.1	Board-level view	5
6	Pulse generator	6
6.1	Implementation	6
6.2	Board-level view	7
7	Memory-mapped peripherals	8
7.1	I ² C to Wishbone bridge	9
7.2	Control and status registers	9
8	Folder Structure	10
9	Getting Around the Code	11
	Appendices	13
A	Memory map	13
A.1	Control and status registers	13
A.1.1	Board ID register	13
A.1.2	Status register	13

List of Figures

1	Block diagram of FPGA firmware	1
2	FPGA clock inputs	2
3	<i>rtm_detector</i> block in CONV-TTL-BLO firmware	3
4	3x2 bicolor LED matrix control	4
5	Pulse generator block	7
6	Board-level view of pulse replication mechanism	8
7	No signal detect block	8
8	Declarative part of VHDL architecture	11
9	Body of VHDL architecture	12

List of Tables

1	Clock domains	2
2	LED state input	5
3	LED state vector connections in the firmware	5
4	CONV-TTL-BLO memory map	13

List of Abbreviations

DAC	Digital-to-Analog Converter
FPGA	Field-Programmable Gate Array
FSM	Finite-State Machine
IC	Integrated Circuit
I ² C	Inter-Integrated Circuit (bus)
PLL	Phase-Locked Loop
SPI	Serial Peripheral Interface
SysMon	(ELMA) System Monitor
VCXO	Voltage-controlled oscillator

1 Introduction

This document details the HDL implemented on the Spartan-6 FPGA on the CONV-TTL-BLO board. The HDL (mostly implemented in VHDL) handles the following aspects of the CONV-TTL-BLO capabilities:

- pulse detection (on pulse rising edge)
- fixed-width pulse generation
- status retrieval via I²C and the ELMA protocol

Figure 1 shows a simplified block diagram of the HDL firmware. Each of the blocks in the figure is presented in following sections.

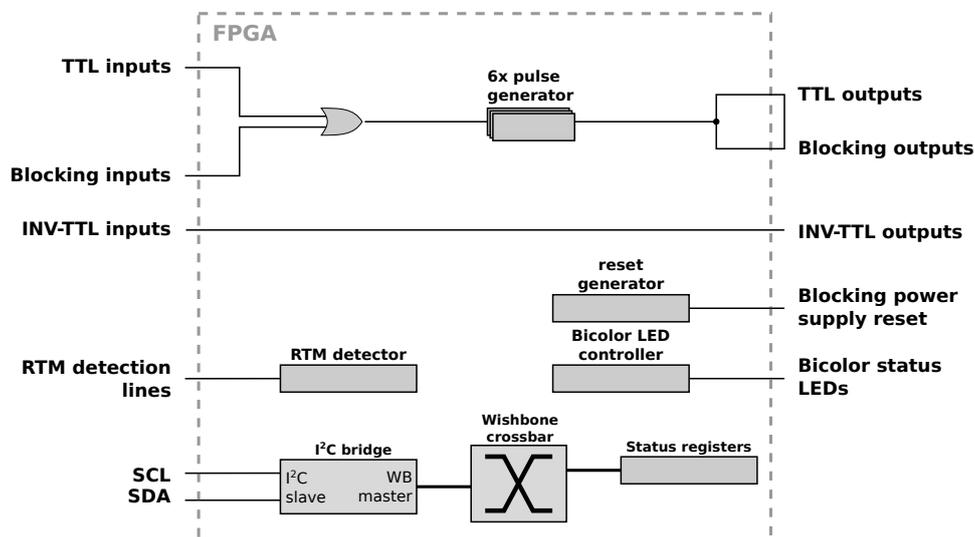


Figure 1: Block diagram of FPGA firmware

Additional documentation

- CONV-TTL-BLO User Guide [1]
- CONV-TTL-BLO Hardware Guide [2]

2 FPGA Clocks

There are two clock signals input to the FPGA (Figure 2). The first is a 20 MHz signal from a VCXO. The second clock signal with a frequency of 125 MHz is generated on-board via a Texas Instruments PLL IC from a 25 MHz VCXO.

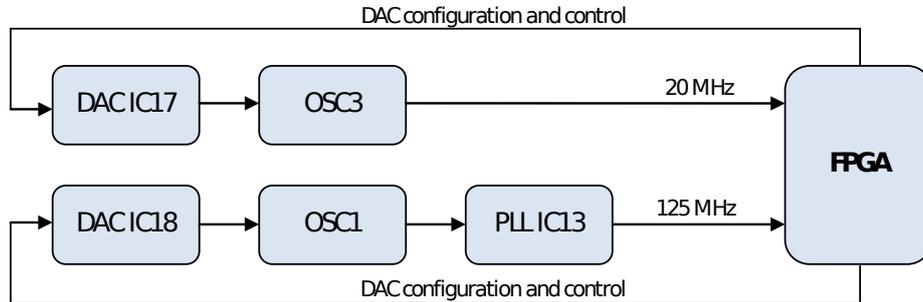


Figure 2: FPGA clock inputs

Two DACs are provided on-board for controlling the two VCXOs. The DACs can be controlled via SPI, but this feature is not yet implemented.

Table 1 lists the clock domains in the firmware.

Table 1: Clock domains

Clock domain	Frequency	Comments
<i>clk125</i>	125 MHz	Global clock input to all sequential logic

3 Reset generator

Entity	<i>reset_gen</i>	
Generics	<i>g_reset_time</i>	Reset time in <i>clk_i</i> cycles
Ports	<i>clk_i</i>	Clock signal
	<i>rst_i</i>	Active-high reset input
	<i>rst_n_o</i>	Active-low reset output
Usage	Global reset generation	96 <i>ms</i> reset

The reset generator module (*reset_gen*) implemented inside the FPGA generates a predefined-width reset signal when power is applied to the FPGA, or when an external reset is triggered via the *rst_i* pin.

When a power-on reset occurs on the Xilinx FPGA, a counter inside the *reset_gen* module starts counting up. While this counter is counting up, the active-low reset signal is kept low, resetting synchronous logic inside the FPGA. When the counter reaches the value of the reset width (specified via the *g_reset_time* generic at synthesis time), the reset signal is de-asserted, the counter is disabled and the *reset_gen* module remains inactive.

The module reactivates on the power-on reset, or when a reset is triggered externally, via the *rst_i* pin.

Note that the VHDL of this module is Xilinx and XST-specific and porting to a different FPGA architecture is not guaranteed to provide the same

results. The *reset_gen* module has an initial value set for the counter signal after power-up, which is guaranteed by XST to be set after the FPGA's GSR signal is de-asserted.

By default, the reset time is set to 96 *ms*.

4 RTM detection

Entity	<i>rtm_detector</i>	
Ports	<i>rtmm_i(2..0)</i>	RTM mainboard detection lines
	<i>rtmp_i(2..0)</i>	RTM piggyback detection lines
	<i>rtmm_ok_o</i>	RTM mainboard present
	<i>rtmp_ok_o</i>	RTM piggyback present
Usage	Light ERR status LED	

RTM detection is described in [3]. Since an RTMM/P missing would mean all *rtmm_i/rtmp_i* lines are all-ones, the *rtm_detector* module sets the *rtmm_ok* and *rtmp_ok* signals low if the *rtmm_i* and *rtmp_i* input signals are respectively all-ones.

The *rtmm_ok* and *rtmp_ok* signals are NANDed together to light the ERR status LED on the CONV-TTL-BLO.

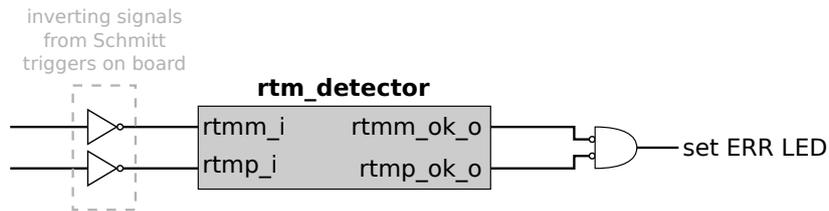


Figure 3: *rtm_detector* block in CONV-TTL-BLO firmware

The status of the RTM detection lines can also be read via their respective fields in the CONV board status register (Appendix A.1).

5 Bicolor LED controller

Entity	<i>bicolor_led_ctrl</i>	
Generics	<i>g_NB_COLUMN</i>	Number of columns
	<i>g_NB_LINE</i>	Number of lines
	<i>g_CLK_FREQ</i>	Frequency (in Hz) of <i>clk_i</i> signal
	<i>g_REFRESH_RATE</i>	LED refresh rate (in Hz)
Ports	<i>rst_n_i</i>	Active-low reset input
	<i>clk_i</i>	Clock signal input
	<i>led_intensity_i(6..0)</i>	7-bit LED intensity vector
	<i>led_state_i(..)</i>	LED state vector, two bits per LED
	<i>column_o(..)</i>	LED column vector, one bit per column
	<i>line_o(..)</i>	LED line vector, one bit per line
	<i>line_oen_o(..)</i>	LED line enable vector, one bit per line
Usage	Light bicolor LEDs	

The *bicolor_led_ctrl* block controls the lighting of a bicolor LED matrix. Based on the refresh rate given via the *g_REFRESH_RATE* generic, the clock frequency (*g_CLK_FREQ* generic) and the number of lines and columns, the module controls lighting each LED in the LED matrix.

Figure 4 shows an example of controlling a three-line, two-column red-and-green LED matrix. The FPGA outputs for the columns (C) are connected to buffers and serial resistors and then to the LEDs. The FPGA outputs for lines (L) are connected to tri-state buffers and then to the LEDs. The FPGA outputs for line output enables (L_OEN) are connected to the output enable of the tri-state buffers.

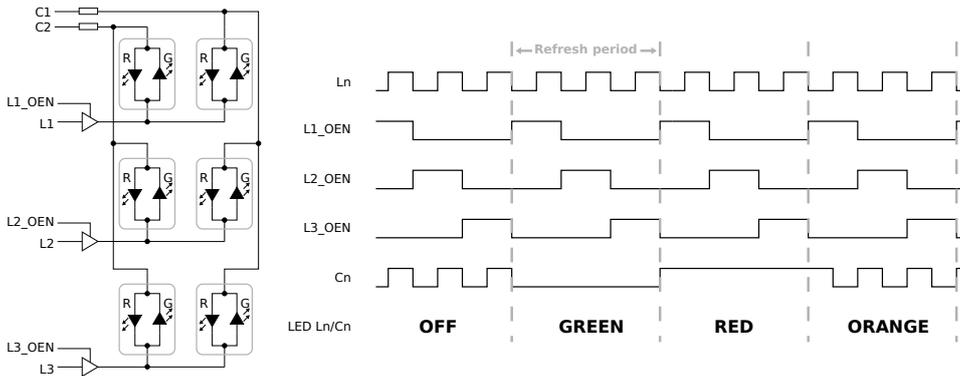


Figure 4: 3x2 bicolor LED matrix control

The two-bit *led_state_i* vector can be used to control the color of each LED. Table 2 lists the values that should be input on *led_state_i* to get the needed color. Definitions are provided in the *bicolor_led_ctrl_pkg.vhd* file for setting the color of the LED via *led_state_i*; these constants are also listed in Table 2.

Table 2: LED state input

State	Constant	Value
Off	c_LED_OFF	00
Green	c_LED_GREEN	01
Red	c_LED_RED	10
Orange	c_LED_RED_ORANGE	11

Each LED's two-bit state is connected to *led_state_i* on a column-first, line-second basis.

5.1 Board-level view

There are twelve bicolor LEDs on the CONV-TTL-BLO; they are connected in a two-line, six-column pattern controlled by a *bicolor_led_ctrl* block. Table 3 shows the *led_state_i* connections for the bicolor status LEDs in the CONV-TTL-BLO firmware.

Table 3: LED state vector connections in the firmware

Line	Column	LED	LED state bits	Setting
1	1	WHITE_RABBIT_ADDR	1..0	
1	2	WHITE_RABBIT_GMT	3..2	
1	3	WHITE_RABBIT_LINK	5..4	
1	4	WHITE_RABBIT_OK	7..6	
1	5	MULTICAST_ADDR_1	9..8	
1	6	MULTICAST_ADDR_2	11..10	
2	1	I2C	13..12	
2	2	TTL	15..14	
2	3	ERR	17..16	
2	4	PW	19..18	
2	5	MULTICAST_ADDR_4	21..20	
2	6	MULTICAST_ADDR_8	23..22	

The states of the used LEDs can be found in Table 1 of [1]. They are controlled by combinatorial multiplexers. The selection signals to these multiplexers are set throughout the logic.

6 Pulse generator

Entity	<i>ctb_pulse_gen</i>	
Generics	<i>g_pwidth</i>	Width of the output pulse in <i>clk_i</i> cycles
	<i>g_gf_len</i>	Length of glitch filter
Ports	<i>clk_i</i>	Clock signal
	<i>rst_n_i</i>	Active-low reset signal
	<i>en_i</i>	Pulse generator enable
	<i>gf_en_n_i</i>	Active-low glitch filter enable
	<i>trig_i</i>	Pulse trigger
	<i>pulse_o</i>	Pulse output
Usage	Output pulse	1.2 μ s pulses with min. period of 6 μ s

The *ctb_pulse_gen* block generates pulses on the rising edge of the *trig_i* input. The pulse width is configurable via the *g_pwidth* generic. The block also incorporates a glitch filter with a configurable length (*g_gf_len*) that can be used to avoid pulses generated because of glitches at the *trig_i* input.

Pulse widths at the output are limited internally to 1/5 duty cycle, to safeguard the blocking output transformers.

Six *ctb_pulse_gen* blocks (one per channel) are used for generating blocking and TTL pulses at the outputs, based on trigger inputs arriving on the channels. The *ctb_pulse_gen* blocks are configured for 1.2 μ s pulses (*g_pwidth* = 150, considering the 8 ns clock input).

6.1 Implementation

Figure 5 shows the implementation of the *ctb_pulse_gen* block. It employs a finite-state machine (FSM) that is used to generate a fixed-width pulse at the output.

The glitch filter can be used to decrease sensitivity to glitches in noisy environments. It can be enabled via the *gf_en_n_i* input (connected to SW1.1 on the CONV-TTL-BLO). The length of the filter can be set via the *g_gf_len* generic.

Enabling the glitch filter will lead to the trigger being sampled using *clk125* and introduces leading-edge jitter on the *pulse_o* output. To avoid this leading-edge pulse jitter, the glitch filter can be left disabled.

Regardless of whether the glitch filter is enabled or not, the FSM reacts to the rising edge of one of its two start inputs. A rising edge on an input starts the internal counter, which counts up to a maximum value of *g_pwidth*. The behavior of the outputs are different, depending on the state of the glitch filter.

With the glitch filter disabled, the input pulse enables the input flip-flop, which starts pulse generation. The pulse signal is then synchronized in the *clk125* domain and input to the synchronous FSM, which extends the pulse

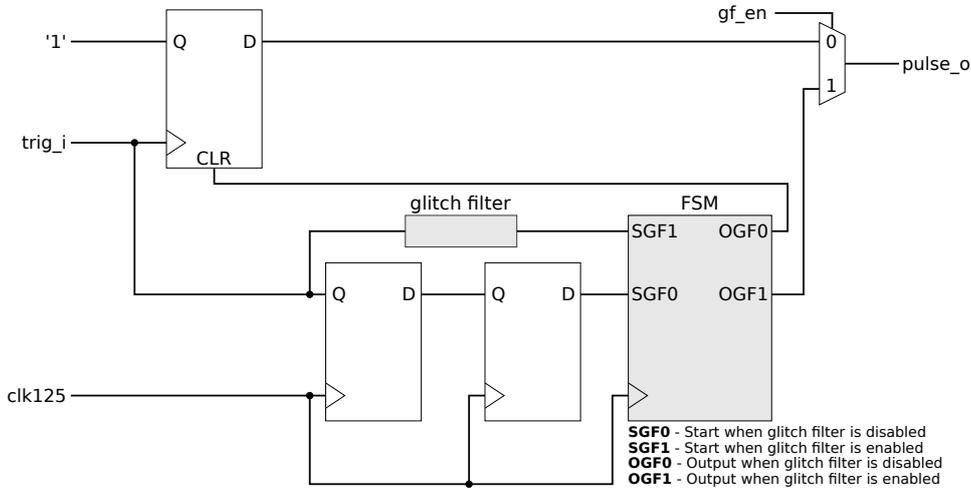


Figure 5: Pulse generator block

to g_pwidth . The rising edge on $SGF0$ triggers the counter, and when the counter reaches the value corresponding to the selected pulse width, it sets the $OGF0$ output, which will reset the input flip-flop, thus ending the pulse.

With the glitch filter enabled, the rising edge on $SGF1$ sets $OGF1$, and this will be kept high until the counter reaches the value corresponding to the pulse width.

After the pulse generation period, the FSM goes into a pulse rejection state, where the pulse reset is kept high. If any pulses arrive on the input while the FSM is in this rejection state, they are not replicated at the output. The pulse rejection phase lasts for $4 * g_pwidth$, yielding a maximum duty cycle of $1/5$ for input pulses.

6.2 Board-level view

Figure 6 shows the pulse replication mechanism on the CONV-TTL-BLO. Here, the PG block is the ctb_pulse_gen block with the necessary settings. Since the ctb_pulse_gen block expects a rising edge at its $trig_i$ input in order to generate a pulse at the output, logic external to the block caters for the different types of signals that arrive on CONV-TTL-BLO inputs.

Most of this external logic is on the TTL pulse side, where both TTL and TTL-BAR pulses may arrive. As described in Section 4.3 of [1], if a wire is not plugged in when TTL-BAR pulses are input, a continuous logic high level on the line would inhibit pulses arriving on the blocking side from triggering a pulse generation. This is why the *no sig. detect* block has been implemented.

The block's implementation is shown in Figure 7. It is implemented as a counter which keeps the en_o signal high as long as it does not reach its

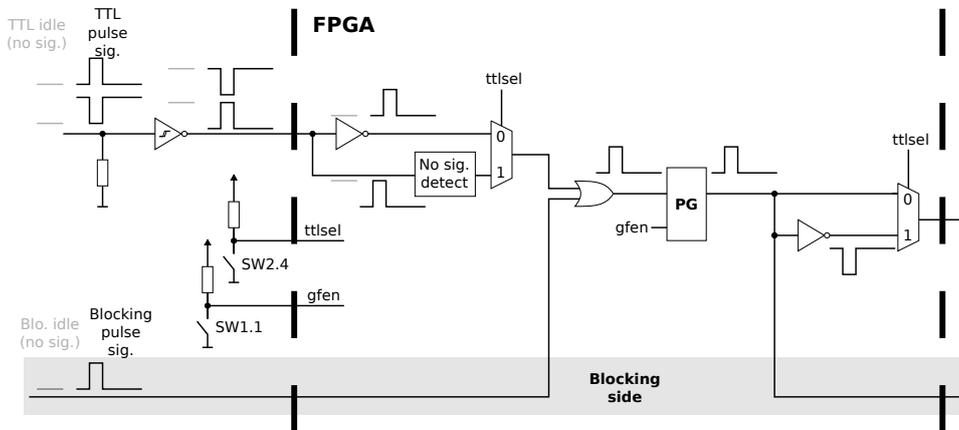


Figure 6: Board-level view of pulse replication mechanism

maximum value. The counter counts up when the *cnt* input is high. By setting the maximum value of the counter to 12499, it disables the line to the multiplexer if this stays high for $100\ \mu\text{s}$, thus allowing for blocking pulses at the input of the OR gate. The line is re-enabled as soon as it goes back low, i.e., when a wire has been plugged in to the channel.

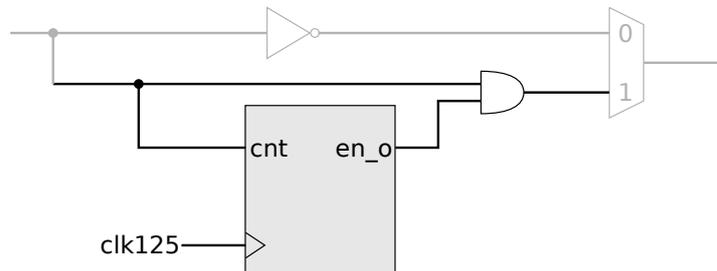


Figure 7: No signal detect block

7 Memory-mapped peripherals

This section details the various peripherals mapped on the internal Wishbone bus. Access to these peripherals is made through the two serial lines on the VME P1 connector (*SERCLK*, *SERDAT*). The I²C-based protocol proposed by ELMA [4] for their VME crates is used to access these peripherals. A bridge module (Section 7.1) translates I²C transfers into Wishbone transfers.

The complete memory map of the firmware can be found in Appendix A.

7.1 I²C to Wishbone bridge

The *elma_i2c* module implements a bridge between the serial lines on the VME P1 connector using the ELMA I²C-based protocol [4], and the Wishbone interconnect. The module provides one I²C slave interface for connecting to an ELMA SysMon and one Wishbone master interface.

Details about the module's implementation can be found in its documentation.

7.2 Control and status registers

The status registers implemented in the firmware contain the current firmware version, the position of the on-board switches and the values on RTM detection lines.

No control registers are currently implemented.

See Appendix A.1 for more information.

8 Folder Structure

The folder structure used within the firmware is presented below.

- ../ip_cores/
- conv-ttl-blo-gw/doc/
- conv-ttl-blo-gw/hdl/
 - bicolor_led_ctrl/
 - *bicolor_led_ctrl.vhd*
 - *bicolor_led_ctrl_pkg.vhd*
 - glitch_filt/
 - rtl/
 - *glitch_filt.vhd*
 - **release/**
 - **rtl/**
 - ***conv_regs.vhd***
 - **top/**
 - ***conv_ttl_blo.vhd***
 - ***conv_ttl_blo.ucf***
 - ctb_pulse_gen/
 - rtl/
 - *ctb_pulse_gen.vhd*
 - reset_gen/
 - rtl/
 - *reset_gen.vhd*
 - elma_i2c/
 - rtl/
 - *i2c_slave.vhd*
 - *elma_i2c.vhd*

The *ip_cores/* folder contains repository files that the firmware uses, such as the Wishbone crossbar (*xwb_crossbar*).

Documentation such as this HDL guide and some HDL modules developed as part of the CONV-TTL-BLO Project can be found in the *conv-ttl-blo/doc/* folder.

Modules that have been developed as part of the CONV-TTL-BLO project are present in their own folders as sub-nodes of the *conv-ttl-blo/hdl/*

folder. In general, the module files are present under an *rtl/* sub-folder. The I²C bridge module folder also contains the instantiated *i2c_slave* module.

The *release/* folder is the main folder in the firmware package, as can be seen from the fact that it is bolded in the folder structure above. It contains top-level files in the *top/* folder (HDL and UCF file for pin definitions) and other specific modules in the *rtl/* folder.

9 Getting Around the Code

As described above, the main part of the release firmware can be found in the *conv-ttl-blo/hdl/release/* folder. The top-level file is *conv-ttl-blo_v2.vhd*.

Ports and signals usually follow the coding guideline at [5]. Most of the top-level ports of the firmware are lower-case versions of their schematics counterparts. The exceptions from this are due to either net names that could not be syntactically represented in VHDL, or net names that have been made clearer in VHDL code. Input ports are assigned to architecture signals and signals are assigned to output ports in each code section.

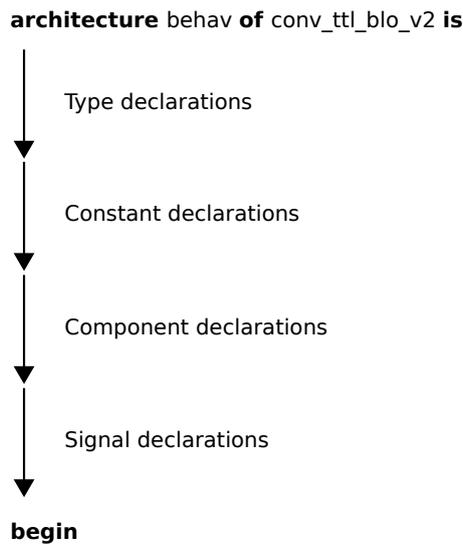


Figure 8: Declarative part of VHDL architecture

The declarative part of the architecture is organized as shown in Figure 8. Types are declared right after the architecture declaration, followed by constant declarations, followed by component declarations, after which the various signals are declared.

The body of the architecture is organised as shown in in Figure 9. It begins by instantiating a differential buffer for the 125 MHz system clock and instantiating the *reset_gen* component. Then, the *elma_i2c* bridge module is instantiated along with the Wishbone crossbar that offers access to

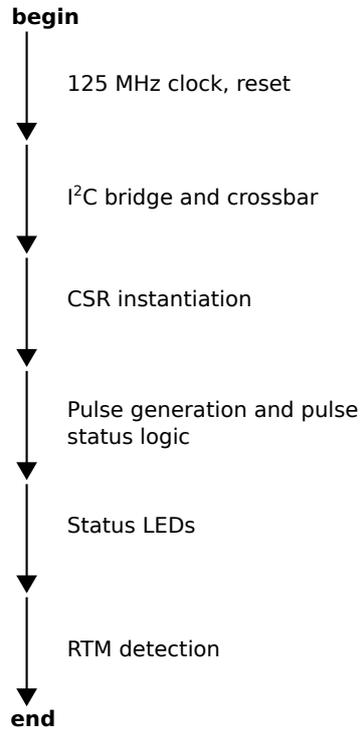


Figure 9: Body of VHDL architecture

the rest of the Wishbone modules in the design. Next, the CONV board CSR module is instantiated, followed by the instantiation of twelve pulse generator modules, six for pulse repetition and six for the pulse LEDs. This is followed by the logic for the status LEDs and the file ends with the RTM detection modules.

Appendices

A Memory map

Table 4 shows the complete memory map of the firmware. The following sections list the memory map of each peripheral.

Table 4: CONV-TTL-BLO memory map

Periph.	Address		Description
	Base	End	
CSR	0x000	0x010	Control and status register

A.1 Control and status registers

Base address: 0x000

Offset	ELMA reg	Description
0x0	1	Board ID register
0x4	2	Status register
0x8	3	Reserved
0xC	4	Reserved

Reserved addresses read undefined and should be written as 0x00000000.

A.1.1 Board ID register

Bits	Field	Access	Default	Description
31..0	<i>id</i>	R/O	0x424c4f32	Board ID

Field	Description
<i>id</i>	Board ID (ASCII string BLO2)

A.1.2 Status register

Bits	Field	Access	Default	Description
15..0	<i>fwvers</i>	R/O	X	Firmware version
23..16	<i>switches</i>	R/O	X	Switch status
29..24	<i>rtm</i>	R/O	X	RTM detection lines
31..30	<i>reserved</i>	R/O	X	

A Memory map

Field	Description
<i>fwvers</i>	Firmware version – leftmost byte <i>hex value</i> is major release <i>decimal value</i> – rightmost byte <i>hex value</i> is minor release <i>decimal value</i> e.g. 0x0101 – v1.01 0x0107 – v1.07 0x0274 – v2.74 etc.
<i>switches</i>	Current switch status bit 0 – SW1.1 bit 1 – SW1.2 ... bit 7 – SW2.4 1 – switch is OFF 0 – switch is ON
<i>rtm</i>	RTM detection lines status 0 – line active 1 – line inactive
<i>reserved</i>	Write as '0'; read undefined

References

- [1] T.-A. Stana, “CONV-TTL-BLO User Guide.” <http://www.ohwr.org/documents/263>, 06 2013.
- [2] T.-A. Stana, “CONV-TTL-BLO Hardware Guide.” <http://www.ohwr.org/documents/282>, 07 2013.
- [3] “Rear Transition Module detection.” http://www.ohwr.org/projects/conv-ttl-blo/wiki/RTM_board_detection.
- [4] ELMA, “Access to board data using SNMP and I2C.” www.ohwr.org/attachments/download/2324/ELMA_SNMP_specification.pdf.
- [5] P. Loschmidt, N. Simanić, C. Prados, P. Alvarez, and J. Serrano, “Guidelines for VHDL Coding,” 04 2011. <http://www.ohwr.org/documents/24>.